Design Review 1

ECE3663 Digital Integrated Circuit

Leiqing Cai (lc9ac)
Qing Qin (qq3za)
Ashley Morse (alm3tw)

3/6/2014

Introduction

The overarching goal of the project is to create a DSP system that consists of three 16-bit registers and an ALU. In this first stage, we implemented some basic components of the ALU, including a 16-bit AND, a 16-bit OR, a 16-bit PASS A and a 1 bit 8:1 MUX.

Block Diagram

The block diagram is shown in Figure 1. Ain and Bin are 16-bit inputs, S is a 3 bit input, and Out is a 16-bit output. Everything other than the inputs/outputs and the register is inside the ALU. The register has to be shown in order to show the NOP operation. The second input of the shifter is 4-bit wide instead of 16-bit, so a splitter was use to retrieve the least significant 4 bits of Bin.

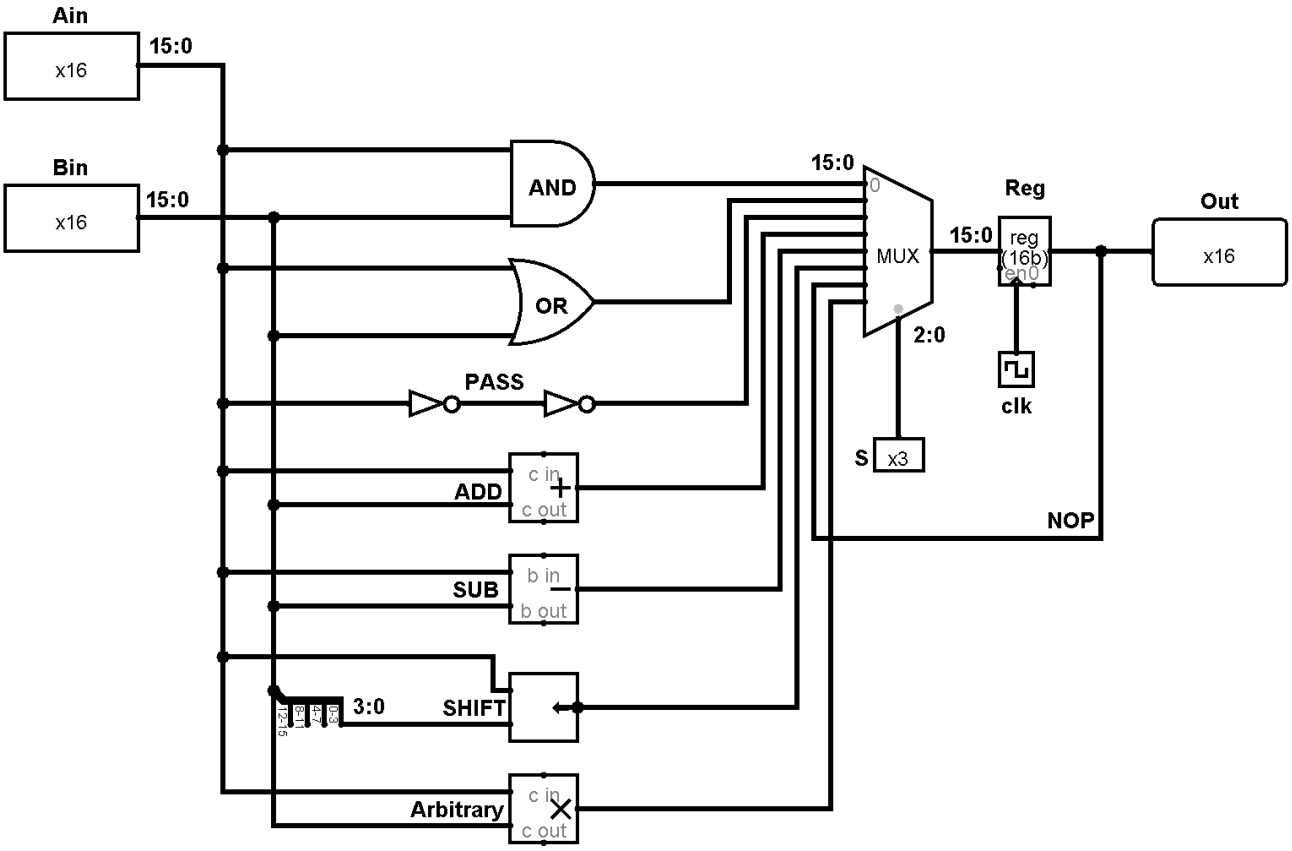

Figure 1 Block Diagram of the ALU

Components

Introduction

We designed our components by directly editing netlists. The reason we did so is that the design can be more clearly specified and finely controlled in a netlist. We uses a bottom-up approach, starting by small and simple components.

Naming Conventions

Most of the sub-circuit we built follows the naming conventions:

$$< Functionality >< fan > \_ < width >$$

Where fan stands for the number of data inputs to the component, and width stands for the number of bits per data input.

Transistors

Although we may directly use PMOS_VTL and NMOS_VTL from the FreePDK 45nm Technology Library, we choose to create "wrapper" NMOS and PMOS transistors to provide convenience for later design. The netlists are shown below. We introduced the parameter *size*, which represents the relative W/L value to a characteristic transistor.

```
subckt NMOS d g s b
parameters size=1 w=90n l=50n mult=1
 MN (d g s b) NMOS_VTL w=w*size l=l as=100n*w*size ad=100n*w*size ps=200n+w*size pd=200n+w*size m=mult
ends NMOS

subckt PMOS d g s b
parameters size=1 w=90n l=50n mult=1
 MP (d g s b) PMOS_VTL w=w*size l=l as=100n*w*size ad=100n*w*size ps=200n+w*size pd=200n+w*size m=mult
ends PMOS
```

Netlist 1 NMOS and PMOS transistors

Inverter

The benefit of introducing parameter *size* has now become apparent. For the NMOS and the PMOS of an inverter, we only need to specify its *size*.

```
subckt Inverter VDD VSS in out
parameters size=1
 MN (out in VSS VSS) NMOS size=size
 MP (out in VDD VDD) PMOS size=size
ends Inverter
```

Netlist 2 Inverter

Buffer

As the inputs for the AND and OR gate are all 16-bit wide, and each of them requires buffering, it is helpful to create a sub-circuit for a buffer. A buffer is simply two inverters in series.

```
subckt Buffer VDD VSS in out
parameters size=1
 I0 (VDD VSS in mid) Inverter size=size
 I1 (VDD VSS mid out) Inverter size=size
ends Buffer
```

Netlist 3 Buffer

16-bit AND Gate

To build a 16-bit AND gate, we first build a 1-bit AND gate and then instantiate 16 copies of it. The 1-bit AND gate is a NAND gate in series with an inverter, thus a total of 6 transistors. The two pull-down NMOS transistors are in series so they are sized to 2. We can achieve this simply by setting the *size* parameter.

```
subckt AND2 VDD VSS A B o
parameters size=1
 // Pull-down network of NOR
 M0 (obar A net0 VSS) NMOS size=2*size
 M1 (net0 B VSS VSS) NMOS size=2*size
 // Pull-up network of NOR
 M2 (obar A VDD VDD) PMOS size=size
 M3 (obar B VDD VDD) PMOS size=size
 // Inverter
 I0 (VDD VSS obar o) Inverter size=size
ends AND2
```

Netlist 4 1-bit AND gate

```
subckt AND2_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0
parameters size=1
 M15 (VDD VSS A15 B15 o15) AND2 size=size
 M14 (VDD VSS A14 B14 o14) AND2 size=size
 M13 (VDD VSS A13 B13 o13) AND2 size=size
 M12 (VDD VSS A12 B12 o12) AND2 size=size
 M11 (VDD VSS A11 B11 o11) AND2 size=size
 M10 (VDD VSS A10 B10 o10) AND2 size=size
 M9 (VDD VSS A9 B9 o9) AND2 size=size
 M8 (VDD VSS A8 B8 o8) AND2 size=size
 M7 (VDD VSS A7 B7 o7) AND2 size=size
 M6 (VDD VSS A6 B6 o6) AND2 size=size
 M5 (VDD VSS A5 B5 o5) AND2 size=size
 M4 (VDD VSS A4 B4 o4) AND2 size=size
 M3 (VDD VSS A3 B3 o3) AND2 size=size
 M2 (VDD VSS A2 B2 o2) AND2 size=size
 M1 (VDD VSS A1 B1 o1) AND2 size=size
 M0 (VDD VSS A0 B0 o0) AND2 size=size
ends AND2_16
```

Netlist 4 16-bit AND gate

Here, we also provide a cadence schematic of the AND gate for reference. Naming of the ports and instances not to be correspond to the netlist. Simulation is based on the netlists.

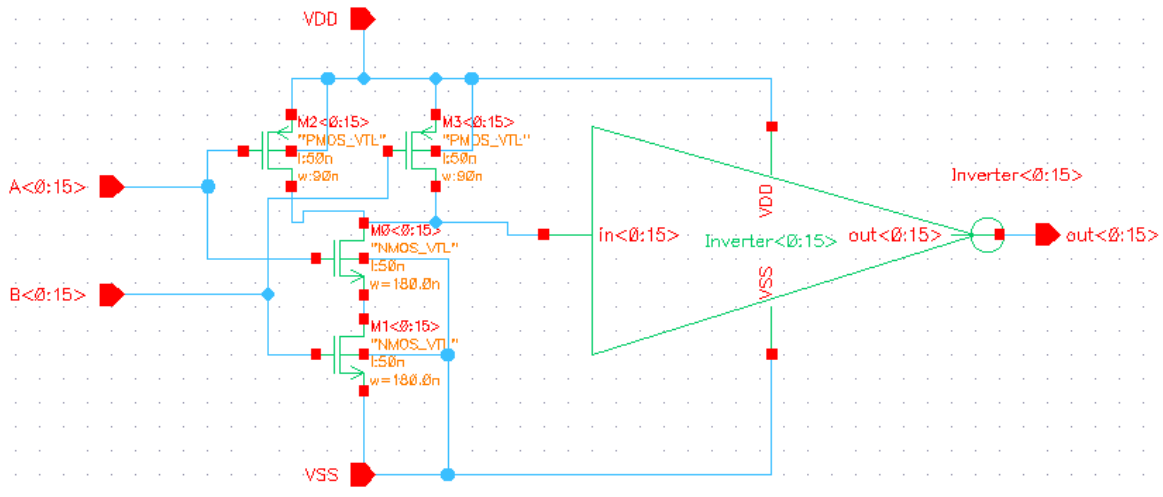

Figure 2 Cadence schematic of AND Gate

16-bit OR Gate

To build a 16-bit OR gate, we first build a 1-bit OR gate and then instantiate 16 copies of it. The 1-bit OR gate is creating by placing an NOR gate in series with an inverter. The pull-up PMOSs are in series so they are sized to 2.

```
subckt OR2 VDD VSS A B o
parameters size=1
 // Pull-down network of NOR
 M0 (obar A VSS VSS) NMOS size=size
 M1 (obar B VSS VSS) NMOS size=size
 // Pull-up network of NOR
 M2 (net0 A VDD VDD) PMOS size=2*size
 M3 (obar B net0 VDD) PMOS size=2*size
 // Inverter
 I0 (VDD VSS obar o) Inverter size=size
ends OR2
```

Netlist 5 1-bit OR gate

```
subckt OR2_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0
parameters size=1
 M15 (VDD VSS A15 B15 o15) OR2 size=size
 M14 (VDD VSS A14 B14 o14) OR2 size=size
 M13 (VDD VSS A13 B13 o13) OR2 size=size
 M12 (VDD VSS A12 B12 o12) OR2 size=size
 M11 (VDD VSS A11 B11 o11) OR2 size=size
 M10 (VDD VSS A10 B10 o10) OR2 size=size
 M9 (VDD VSS A9 B9 o9) OR2 size=size
 M8 (VDD VSS A8 B8 o8) OR2 size=size
 M7 (VDD VSS A7 B7 o7) OR2 size=size
 M6 (VDD VSS A6 B6 o6) OR2 size=size
 M5 (VDD VSS A5 B5 o5) OR2 size=size
 M4 (VDD VSS A4 B4 o4) OR2 size=size
 M3 (VDD VSS A3 B3 o3) OR2 size=size
 M2 (VDD VSS A2 B2 o2) OR2 size=size
 M1 (VDD VSS A1 B1 o1) OR2 size=size
 M0 (VDD VSS A0 B0 o0) OR2 size=size
ends OR2_16
```

Netlist 6 16-bit OR gate

16-bit Pass A

Pass A can either be just a wire for each bit, or alternatively a buffer for each bit. We decided to choose the latter. The 1-bit Pass A is the same as a buffer.

```
subckt PASS VDD VSS A o
parameters size=1
 Buf (VDD VSS A o) Buffer size=size
ends PASS
```

Netlist 7 1-bit PASS A

```
subckt PASS_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0
parameters size=1
 M15 (VDD VSS A15 o15) PASS size=size
 M14 (VDD VSS A14 o14) PASS size=size
 M13 (VDD VSS A13 o13) PASS size=size
 M12 (VDD VSS A12 o12) PASS size=size
 M11 (VDD VSS A11 o11) PASS size=size
 M10 (VDD VSS A10 o10) PASS size=size
 M9 (VDD VSS A9 o9) PASS size=size
 M8 (VDD VSS A8 o8) PASS size=size
 M7 (VDD VSS A7 o7) PASS size=size
 M6 (VDD VSS A6 o6) PASS size=size
 M5 (VDD VSS A5 o5) PASS size=size
 M4 (VDD VSS A4 o4) PASS size=size
 M3 (VDD VSS A3 o3) PASS size=size
 M2 (VDD VSS A2 o2) PASS size=size
 M1 (VDD VSS A1 o1) PASS size=size
 M0 (VDD VSS A0 o0) PASS size=size
ends PASS_16
```

Netlist 8 16-bit PASS A

8-to-1 MUX

We first built a 2-to-1 MUX, and then uses 7 instances of it to build the 8-to-1 MUX.

```
subckt MUX2 VDD VSS A B S out
parameters size=1
 // Pull-Down Network
 M0 (net1 B net3 VSS) NMOS size=2*size
 M1 (net3 S VSS VSS) NMOS size=2*size
 M2 (net1 A net4 VSS) NMOS size=2*size
 M3 (net4 net2 VSS VSS) NMOS size=2*size

 // Pull-up Network
 M4 (net1 A net5 VDD) PMOS size=2*size
 M5 (net5 S VDD VDD) PMOS size=2*size
 M6 (net1 net2 net5 VDD) PMOS size=2*size
 M7 (net5 B VDD VDD) PMOS size=2*size

 // Inverter
 I0 (VDD VSS S net2) Inverter size=size
 I1 (VDD VSS net1 out) Inverter size=size
ends MUX2
```

Netlist 9 2-to-1 MUX

```
subckt MUX8 VDD VSS in7 in6 in5 in4 in3 in2 in1 in0 s2 s1 s0 out
parameters size=1
 M0 (VDD VSS in0 in4 s2 net0) MUX2 size=size
 M1 (VDD VSS in2 in6 s2 net1) MUX2 size=size
 M2 (VDD VSS in1 in5 s2 net2) MUX2 size=size
 M3 (VDD VSS in3 in7 s2 net3) MUX2 size=size
 M4 (VDD VSS net0 net1 s1 net4) MUX2 size=size
 M5 (VDD VSS net2 net3 s1 net5) MUX2 size=size
 M6 (VDD VSS net4 net5 s0 out) MUX2 size=size
ends MUX8
```

Netlist 10 8-to-1 MUX

Design Verification

16-bit AND gate

Since there are too many combinations of inputs (2^{32}), it is impossible for us to enumerate all the possible combinations. However, due to the fact that each bit location (A_i, B_i, o_i) is constructed identically using a 1-bit AND gate, it is reasonably to show the functionally of just 1 bit. Figure 4 shows a plot from the simulation. Each smallest block is 1ns period, and 4ns of simulation enumerates all the 4 combinations of inputs at bit location 7. For a AND gate, the output is high only when both inputs are high. Based on the left and the middle plot for the inputs, only the first 1ns of the output should give high output and the remaining 3ns should be low. This result thus verifies the correctness of the logic function.

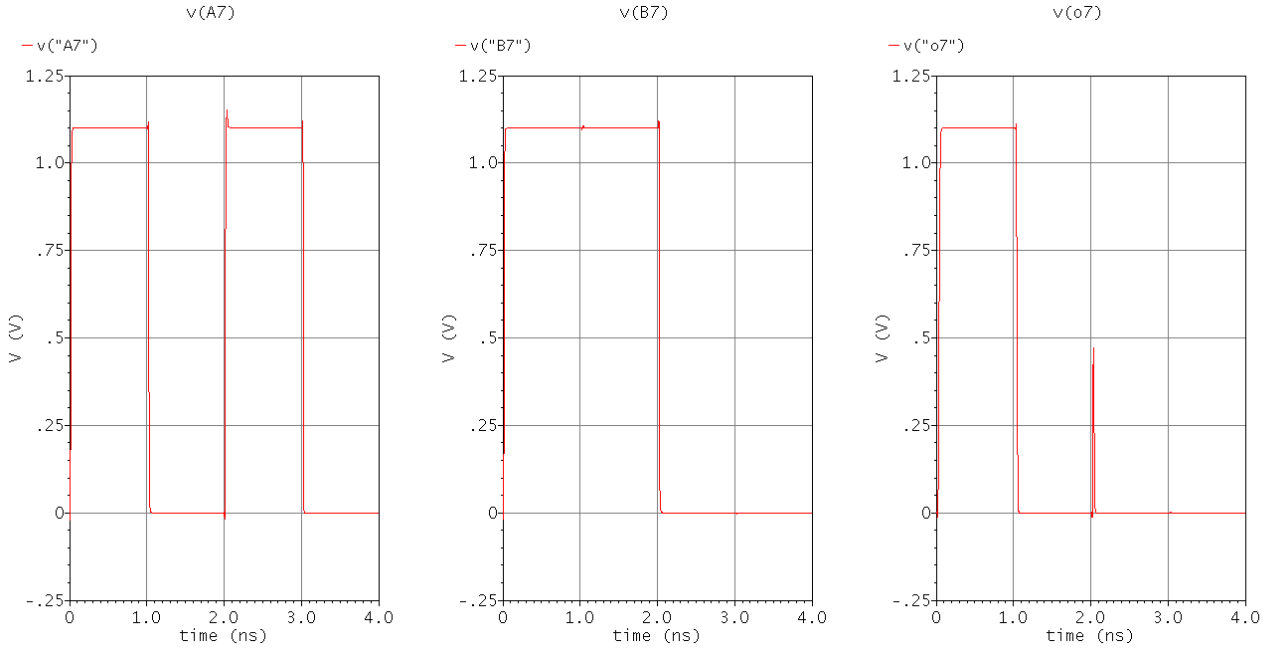

Figure 3 Simulation plot for 16-bit AND gate @ Bit 7 - 4ns (left: A7, middle: B7, right o7)

In order to further improve our confidence of the correctness of our design. We used Ocean to print out the voltage values of the inputs and outputs at $time = 0.5ns, 1.5ns, 2.5ns, 3.5ns$ at all bit locations. These time instance are chosen because they are the middles of the 1ns cycles and so the outputs are steady. A sample of the output file we produced using Ocean is shown below. This basically confirmed that every bit location is free of error (i.e. typo in the netlist).

Bit 0:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 1:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 2:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 3:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 4:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 5:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 6:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 7:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 8:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 9:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 10:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 11:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 12:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 13:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 14:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

Bit 15:

t	v(A)	v(B)	v(o)
0.5n	1.100	1.100	1.100
1.5n	0.000	1.100	0.000
2.5n	1.100	0.000	0.000
3.5n	0.000	0.000	0.000

File 1 Output of AND gate simulation produced using Ocean

16-bit OR gate

Simulation of the OR gate follows the same step as the AND gate. Figure 4 shows the plot. Since the output of an OR gate is low only when both inputs are low, the last 1ns period of the output should be low and the first 3ns should be high. Figure 4 thus verifies its correctness. For the OR gate, we used the method described in the last section to verify each the correctness of each bit of the OR gate as well. The file looks similar but with the correct input/output relations for an OR gate, and so we list its content to avoid being lengthy.

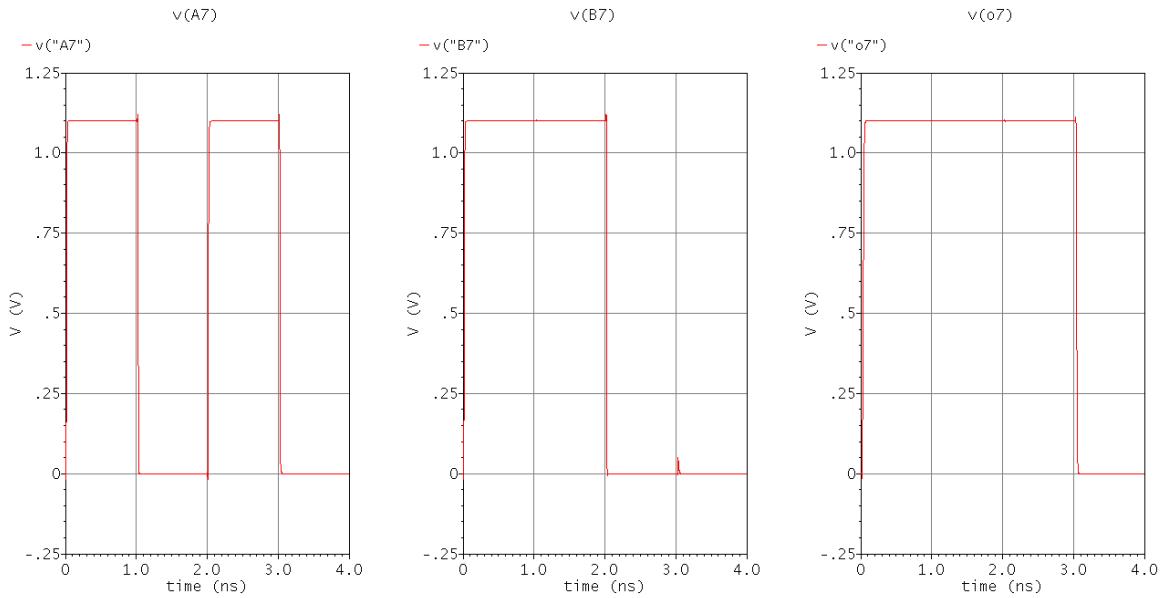

Figure 4 Simulation plot for 16-bit AND gate @ Bit 7 - 4ns (left: A7, middle: B7, right o7)

PASS A

The correctness of PASS A is easy to verify. The output is supposed to look similar to the input.

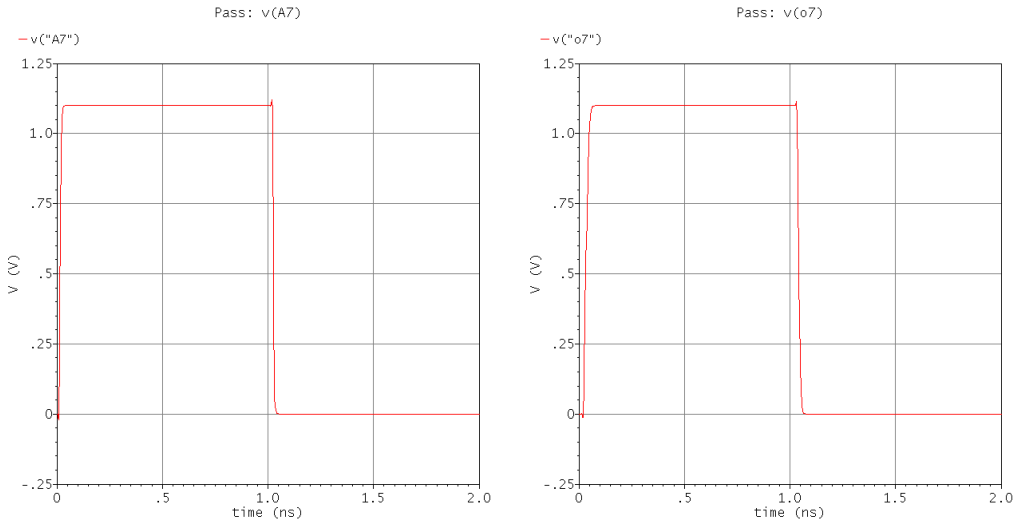

Figure 5 Simulation plot for 16-bit AND gate @ Bit 7 - 2ns (left: A7, right: o7)

8-to-1 MUX

The multiplexer is verify in 2 different ways. First, as it only consists of a bunch of 2-to-1 mux, we simulate would like to verify the smaller-sized mux first. Figure 6 and 7 shows this simulation result. The arrangement of input signals enumerate all possible combinations.

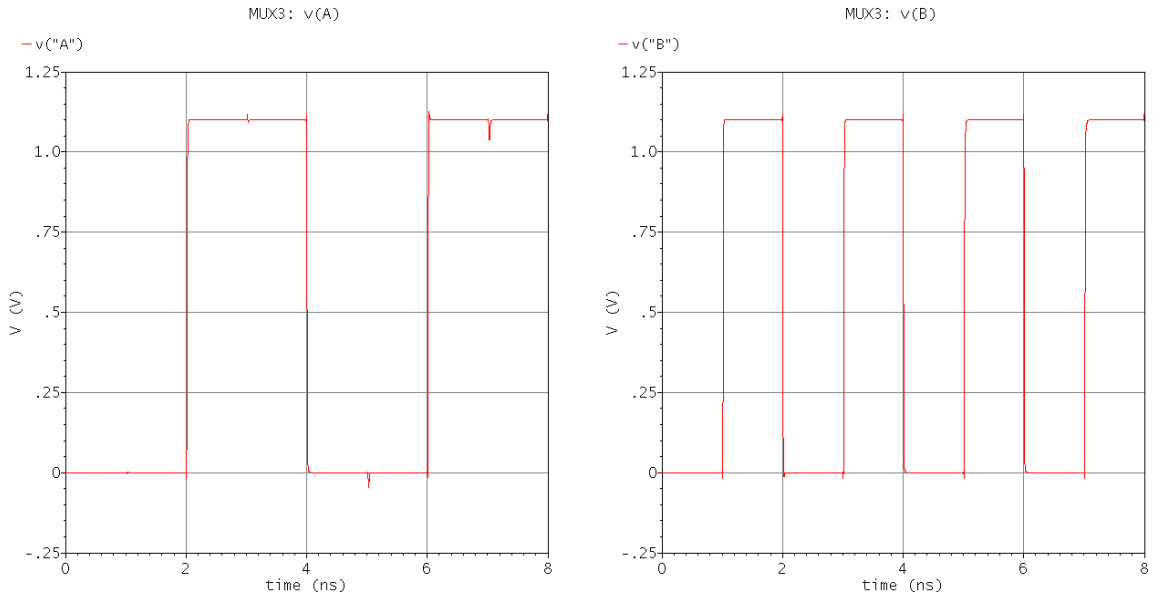

Figure 6 Simulation plot for 2-to-1 mux - 8ns (left: A, right: B)

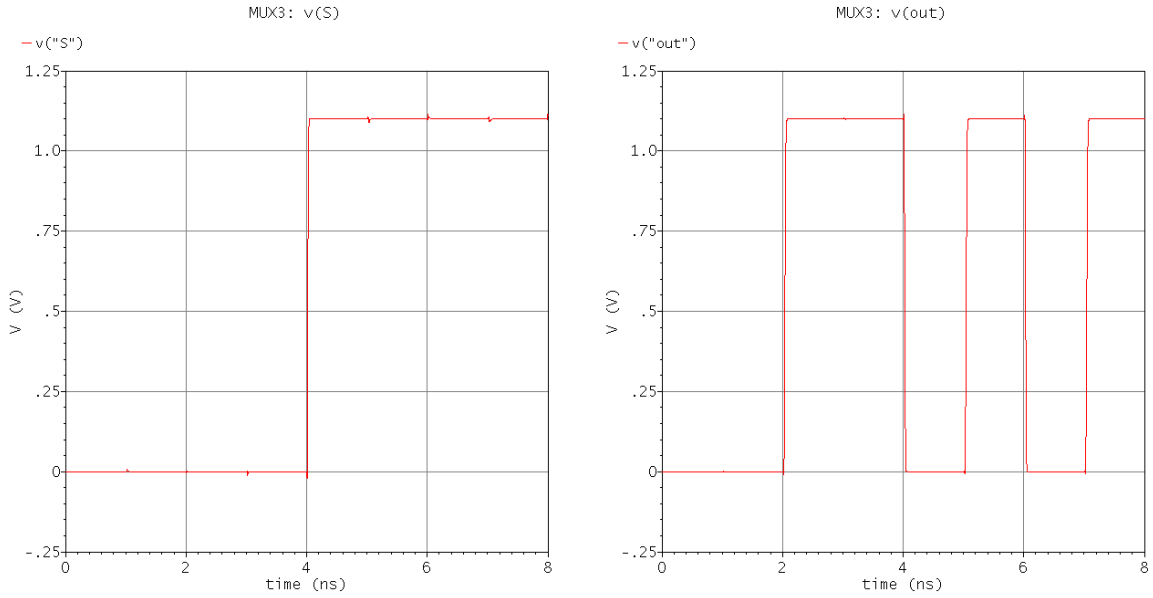

Figure 7 Simulation plot for 2-to-1 mux - 8ns (left: S, right: out)

Base on the input values, the output should look like input A for the first 4ns, and then change to look like B in the remaining 4ns. The plot of $v(out)$ verifies such.

The second run is to test the 8-to-1 mux as a whole. Since we won't be able to test any combination of the inputs. We decided to assign the data inputs ($in7$ - $in0$) a value $in = 01101101_2$(randomly selected), and we swept through all the 8 possible combinations of the selection bits. Figure 8 and 9 shows the plot. The select bits swept from 0 to 7 in the 8ns (1ns for each value). The output should therefore be 10110100_2 (counting in from LSB to MST), and the last plot verifies this.

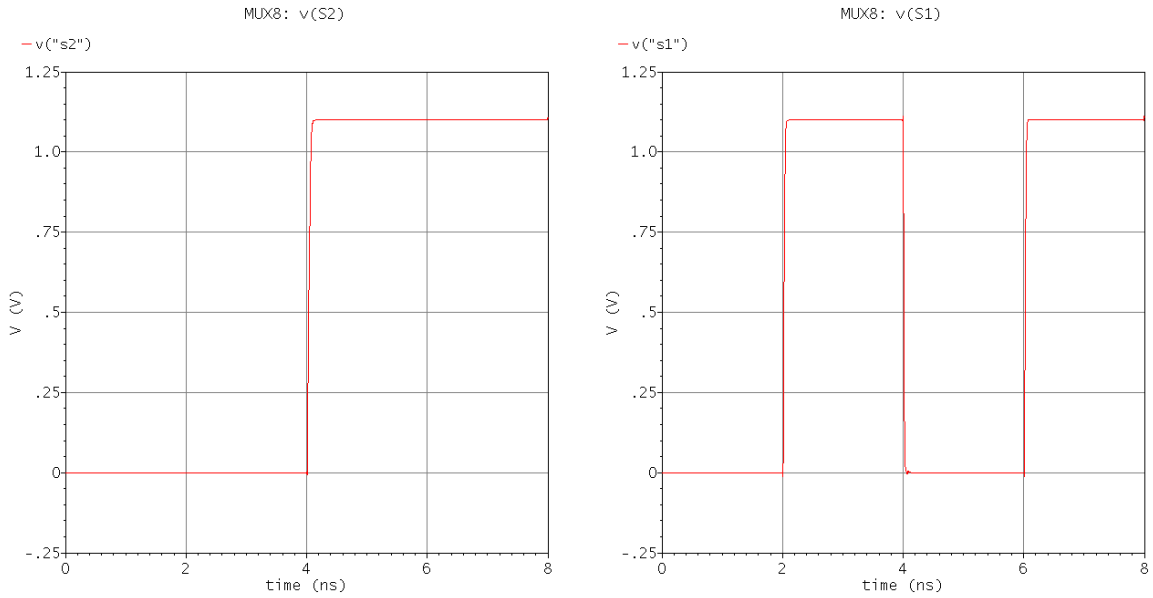

Figure 8 Simulation plot for 8-to-1 mux - 8ns (left: S2, right: S1)

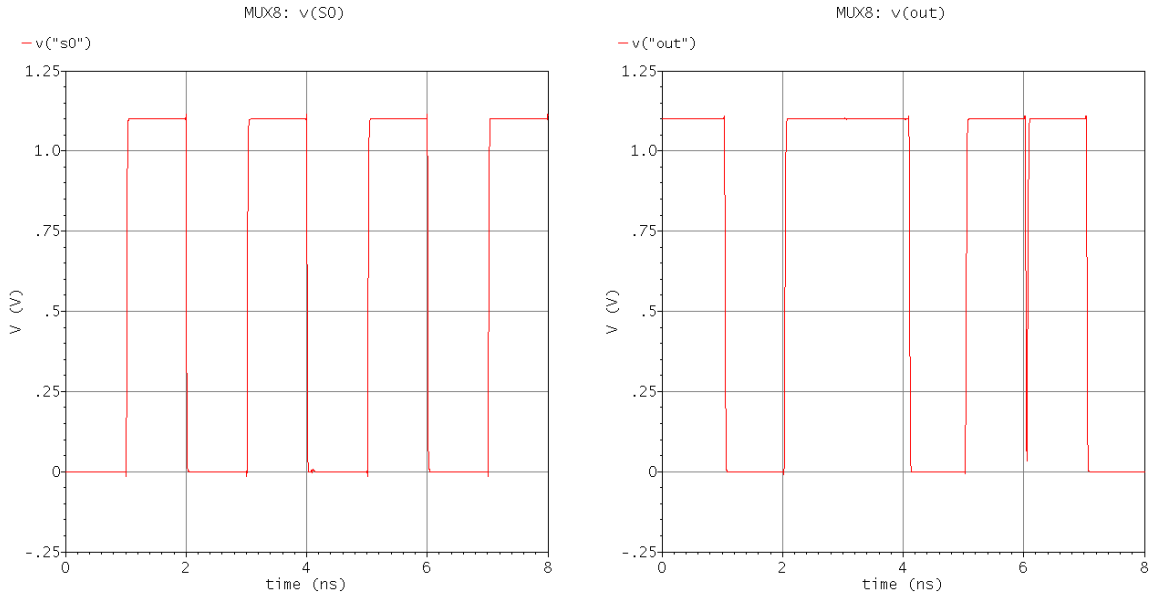

Figure 9 Simulation plot for 8-to-1 mux - 8ns (left: S2, right: S1)

Progress

Tasks Accomplished

In this design review, we created circuits for 16-bit AND, OR, PASS A, and 8:1 MUX and demonstrated that they have been implemented correctly. In addition, we use a 16-bit 8:1 mux that allowed us to select the output from any of the AND, OR and the PASS A gate. We wrote netlists with the FreePDK 45nm technology library. They were then simulated in Spectre with the assistance of Ocean. We managed to demonstrate and show clearly the functionality of the components we designed, without considering optimizing their performance.

Tasks Left - Before Design Review II

During the second phase of the project, our group will first focus on designing and demonstrating the functionality of the components listed below.

- 16-Bit ADD
- 16-Bit SUB
- 16-Bit SHIFT
- ALU in/out connectivity
- NOP
- Register
- Additional functionality

After proving the functionality of the components, our group will move on working on improving the performance of each component based on the metric, $active\ power * delay^2 * area$. Our group will also try doing more research and seeking advice from Prof. Calhoun to push the project forward. The group's wiki page will be updated along the progresses.

Tasks Left: Before Final Presentation

During the last phase of the project, our group will work on putting pieces together a complete design and optimizing the overall performance, balancing the tradeoffs of turning key knobs. After the design is finalized, our group will prepare the conference-style final report required and the final presentation.

Ideas for Arbitrary Function

Some ideas for the arbitrary function that our group will implement are listed below.

- 16-Bit Multiplier
- 16-Bit Maximum Finder